Model Development Phase

Date	2 October 2024
Team ID	SWTID1726888137
Project Title	intelligent handwritten digit identification system for computer applications
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Input
from keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="keras")
```

```
# Load dataset directly from the Keras library
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
# Plot the first six samples of the MNIST training dataset as grayscale images
for i in range(6):
    plt.subplot(230 + i + 1)
    plt.imshow(X_train[i], cmap=plt.get_cmap('gray'))
    plt.axis('off')  # Hide axis ticks
plt.show()
```

```
# Reshape format [samples][width][height][channels]
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

```
# Convert class vectors (integers) to binary class matrix
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False
)
datagen.fit(X_train)
```

```
# Define a CNN model with Input layer
def create_model():
    num_classes = 10
    model = Sequential()

    # Add an Input layer
    model.add(Input(shape=(28, 28, 1)))  # Specify the input shape here

    # Convolutional layers
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # Flatten and fully connected layers
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model

# Build the model
model = create_model()
```

```
# Fit the model with data augmentation
model.fit(datagen.flow(X_train, y_train, batch_size=200),
          validation_data=(X_test, y_test),
          epochs=10, verbose=2)

print("The model has successfully trained.")
```

```
# Save the model
model.save('model.keras')
print("The model has successfully saved.")
```

```
# Evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("CNN error: %.2f%%" % (100 - scores[1] * 100))
```

Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics
Model 1	Model: "sequential" _____ Layer (type) Output Shape Param # ========== conv2d (Conv2D) (None, 26, 26, 32) 320 batch_normalization (BatchN ormalization) (None, 26, 26, 32) 128 conv2d_1 (Conv2D) (None, 24, 24, 64) 18496 batch_normalization_1 (Batc hNormalization) (None, 24, 24, 64) 256 max_pooling2d (MaxPooling2D) (None, 12, 12, 64) 0 dropout (Dropout) (None, 12, 12, 64) 0 flatten (Flatten) (None, 9216) 0 dense (Dense) (None, 256) 2359296 batch_normalization_2 (Batc hNormalization) (None, 256) 1024 dropout_1 (Dropout) (None, 256) 0 dense_1 (Dense) (None, 10) 2570 ========== Total params: 2,377,090 Trainable params: 2,376,706 Non-trainable params: 384 _____	Epoch 1/10 300/300 - 52s - 175ms/step - accuracy: 0.9018 - loss: 0.3262 - val_accuracy: 0.1913 - val_loss: 6.4387 Epoch 2/10 300/300 - 51s - 169ms/step - accuracy: 0.9625 - loss: 0.1201 - val_accuracy: 0.9112 - val_loss: 0.2661 Epoch 3/10 300/300 - 53s - 178ms/step - accuracy: 0.9718 - loss: 0.0935 - val_accuracy: 0.9874 - val_loss: 0.0429 Epoch 4/10 300/300 - 53s - 178ms/step - accuracy: 0.9765 - loss: 0.0794 - val_accuracy: 0.9892 - val_loss: 0.0342 Epoch 5/10 300/300 - 53s - 176ms/step - accuracy: 0.9782 - loss: 0.0713 - val_accuracy: 0.9874 - val_loss: 0.0439 Epoch 6/10 300/300 - 52s - 175ms/step - accuracy: 0.9792 - loss: 0.0674 - val_accuracy: 0.9847 - val_loss: 0.0469 Epoch 7/10 300/300 - 53s - 176ms/step - accuracy: 0.9800 - loss: 0.0635 - val_accuracy: 0.9894 - val_loss: 0.0319 Epoch 8/10 300/300 - 52s - 174ms/step - accuracy: 0.9819 - loss: 0.0597 - val_accuracy: 0.9786 - val_loss: 0.0732 Epoch 9/10 300/300 - 52s - 175ms/step - accuracy: 0.9825 - loss: 0.0562 - val_accuracy: 0.9857 - val_loss: 0.0447 Epoch 10/10 300/300 - 53s - 176ms/step - accuracy: 0.9840 - loss: 0.0530 - val_accuracy: 0.9929 - val_loss: 0.0230 The model has successfully trained.